AIAA-93-0672

A NEW PROCEDURE FOR DYNAMIC ADAPTION OF THREE-DIMENSIONAL UNSTRUCTURED GRIDS

Rupak Biswas

RIACS, Mail Stop T045-1
NASA Ames Research Center
Moffett Field, CA 94035

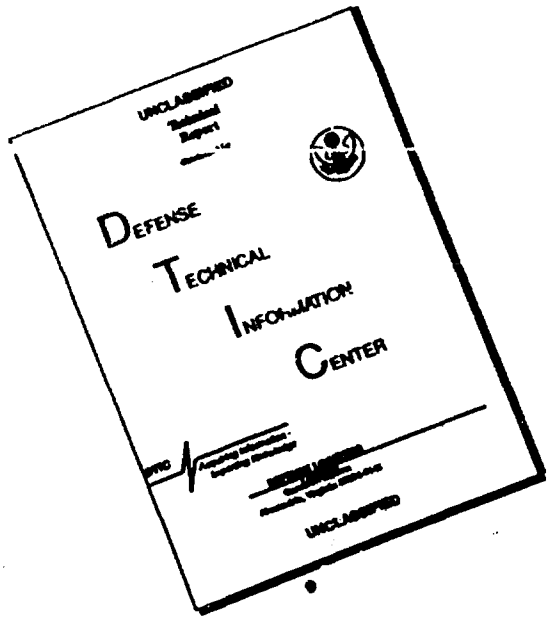
Roger Strawn

US Army AFDD, ATCOM, Mail Stop N258-1
NASA Ames Research Center
Moffett Field, CA 94035

31st Aerospace Sciences Meeting & Exhibit

January 11-14, 1993 / Reno, NV

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

A NEW PROCEDURE FOR DYNAMIC ADAPTION OF THREE-DIMENSIONAL UNSTRUCTURED GRIDS

Rupak Biswas[1]
RIACS, Mail Stop T045-1
NASA Ames Research Center, Moffett Field, CA 94035

Roger Strawn[2]
US Army AFDD, ATCOM, Mail Stop N258-1
NASA Ames Research Center, Moffett Field, CA 94035

ABSTRACT

A new procedure is presented for the simultaneous coarsening and refinement of three-dimensional unstructured tetrahedral meshes. This algorithm allows for localized grid adaption that is used to capture aerodynamic flow features such as vortices and shock waves in helicopter flowfield simulations. The mesh-adaption algorithm is implemented in the C programming language and uses a data structure consisting of a series of dynamically-allocated linked lists. These lists allow the mesh connectivity to be rapidly reconstructed when individual mesh points are added and/or deleted. The algorithm allows the mesh to change in an anisotropic manner in order to efficiently resolve directional flow features. The procedure has been successfully implemented on a single processor of a Cray Y-MP computer. Two sample cases are presented involving three-dimensional transonic flow. Computed results show good agreement with conventional structured-grid solutions for the Euler equations.

INTRODUCTION

Unstructured grids for solving problems in computational fluid dynamics have two major advantages over their structured-grid counterparts. First, the unstructured mesh allows for fast and efficient grid generation around highly complex geometries. Second, appropriate unstructured-grid data structures facilitate the insertion and deletion of points and enable the computational mesh to locally adapt to the flowfield solution.

The work in this paper aims to solve problems in rotary-wing aerodynamics. Here the rotor blade surface geometries are fairly simple, but the resulting flow features are extremely complex. Details of helicopter rotor wakes and acoustics require highly localized regions of mesh refinement in order to accurately predict rotor airloads, vibrations, and noise. The solution-adaptive capabilities of unstructured grids are, therefore, very important for these problems.

Two types of solution-adaptive grid strategies have recently been used with unstructured-grid methods. The first is a grid regeneration scheme where an initial solution is obtained on a coarse mesh and then some error indicator is used to designate regions in the flowfield where additional grid points are required. The grid is then regenerated with a higher clustering of grid points in the targeted flow regions. One major disadvantage of this scheme is that the mesh must be frequently adapted for unsteady flows which is a computationally intensive procedure. However, an advantage of this scheme is that the resulting grids are usually well-formed with smooth transitions between regions of coarse and fine mesh spacing.

A second strategy for producing solution-adaptive meshes involves local modification of the existing grid in regions where the solution is either changing rapidly or is relatively constant. New grid points are individually added to the existing grid in regions where the error indicator is high, and removed from regions where the error indicator is low. The advantage of this strategy is that relatively few mesh points need to be deleted or added at each coarsening/refinement step. However, the scheme has the disadvantage that complicated logic and data structures are required to keep track of the points that are added and removed. Because of the importance of flowfield unsteadiness in rotorcraft problems, we have chosen the local grid modification scheme as the basis for our dynamic mesh adaption.

The key to the success of dynamic grid adaption

[1] Visiting Scientist
[2] Research Scientist, Member AIAA
[3] This paper is declared the work of the U.S. Government and is not subject to copyright protection in the United States.

is the ability to efficiently delete nodes from and add nodes to the mesh. For an unsteady flow, this coarsening/refinement step must be completed every few time steps, so its efficiency must be comparable to that of the flow solver.

For two-dimensional problems, several authors have presented successful schemes for dynamic mesh adaption while solving the Euler equations [1–4]; however, the method is far more difficult in three dimensions. One reason is that an edge in two dimensions can be shared by at most two triangles but an edge in three dimensions can be shared by several tetrahedra. An edge is defined as a line segment that connects two nodes. Thus, a tetrahedral element contains six edges. This requires that data structures be much more complex for three-dimensional problems.

In spite of these added difficulties, dynamic mesh-adaption schemes have been developed for three-dimensional problems [5–7]; however, the data structures are not described in detail. Computer time and memory requirements are the key factors for evaluating any of these schemes. This is particularly true for large three-dimensional problems on advanced computer architectures.

This paper presents a new procedure for dynamic mesh adaption of three-dimensional tetrahedral grids. The method utilizes computer resources very efficiently due to its innovative data structure that is well-suited for large-scale computations. This data structure consists of dynamically-allocated linked lists that are implemented using the C programming language. It is based on edges of the mesh rather than the tetrahedral elements themselves. This allows for anisotropic mesh refinement and coarsening that results in a more efficient distribution of grid points.

A successful adaptive mesh scheme consists of three components: a flow solver, a strategy for identifying regions for refinement and coarsening, and a mechanism for dynamically altering the mesh. The next two sections of this paper will briefly describe an unstructured-grid Euler solver and a criterion for targeting regions for coarsening and refinement. The new data structure and mesh-adaption algorithm will then be described in detail. Particular attention has been given to the scaling of the data structure and associated computer resources requirements for large three-dimensional problems. In addition, two examples will be presented involving multiple levels of solution-adaptive mesh refinement and coarsening for transonic problems.

EULER FLOW SOLVER

The computer code used for testing and validating the dynamic mesh-adaption scheme is a modified version of the three-dimensional Euler solver developed by Barth [8]. The finite-volume upwind scheme solves for solution variables at the vertices of the mesh and satisfies the integral conservation laws on non-overlapping polyhedral control volumes surrounding these vertices. It is a faithful implementation of Godunov's upwind scheme on generalized unstructured meshes. Improved solution accuracy is achieved by using a piecewise linear reconstruction of the solution in each control volume. This improved spatial accuracy hinges heavily on the calculation of the solution gradient in each control volume given pointwise values of the solution at the vertices of the mesh. The solution is advanced in time using conventional explicit procedures.

A rotary-wing version of this code was developed by Strawn and Barth [9]. The governing Euler equations have been rewritten in an inertial reference frame so that the rotor blade and grid system move through stationary air at the specified rotational and translational speeds. Fluxes across each computational control volume were computed using the relative velocities between the moving grid and the stationary far field. This formulation is valid for rotors in hover and forward flight.

An important highlight is that the code uses an edge-based data structure rather than an element-based one. Since the number of edges in a mesh is significantly smaller than the number of faces, cell-vertex edge schemes are inherently more efficient than cell-centered element methods [8]. Furthermore, an edge-based data structure does not limit the user to a particular volume element. Even though tetrahedral elements are used in this paper, any arbitrary combination of polyhedra can be used.

MESH-ADAPTION CRITERIA

A mesh-adaption strategy consists of two components. The first is the choice of an error indicator for each local region of the mesh. Typically, this error indicator is not a true estimate of the error in the solution; rather, it is an indicator of high gradients in the flowfield that are assumed to be regions of high error. The second component is the choice of the number of mesh points that are added and/or deleted at each adaption step. Both these components have a major impact on the final mesh-adapted solution. The intent of this paper is to use fairly conventional error indicators and adaption strategies in order to demonstrate the new coarsening and refinement algorithm.

Kallinderis et al. [5] used differences in velocity magnitude across each edge of the mesh in order to determine an error indicator for the flowfield. This

approach suffers from the problem that the velocity difference across a shock is constant. Thus, a region near a shock will have a constant error value that is independent of the number of times the grid is refined. This observation was clearly illustrated by Warren et al. [10] who proposed a different error indicator in order to remedy this problem. They chose an error indicator that is equal to the difference in the velocity magnitude across each cell multiplied by a length scale for each element as

$$|E_i| = \left(\frac{l_i}{l_r}\right)^{\frac{1}{r}} |\Delta q_i| \qquad (1)$$

where l_i is a characteristic length for each element, l_r is a reference length which is identical for all cells, and Δq_i is the difference in the velocity magnitude across each cell in the mesh. The presence of the length scale in Eq. (1) means that the error indicator is reduced each time the element is refined. This is true even across a shock. This error indicator is also shown to target more elements for refinement that are located away from a shock which can sometimes dramatically improve the overall results.

Warren et al. [10] applied the above error indicator to mesh adaption for two-dimensional problems using isotropic refinement of each element. In three dimensions, an efficient distribution of mesh points requires anisotropic element refinement. We modified the error indicator in Eq. (1) so that it could be applied to an edge-based data structure and also permit anisotropic refinement. Our error indicator is given as

$$|E_e| = ||\Delta\vec{x} \cdot \Delta\vec{v}|| \qquad (2)$$

where $\Delta\vec{x}$ represents the vector displacement along each edge of the mesh and $\Delta\vec{v}$ represents the change in the velocity vector along each edge. Each edge of the mesh is assigned its own error indicator. Note that this error indicator includes a length scale that allows it to decrease as the mesh is refined even across shocks. It also allows anisotropic mesh refinement since the error indicator is high only when the velocity gradients are oriented along an edge. The error estimate in Eq. (2) is used for the sample cases presented in this paper.

One strategy for choosing the number of mesh points to be removed and added at each adaption step is to set absolute error estimate levels for coarsening and refinement. This is the strategy used in [1,3–6] where elements with error values above a certain threshold are refined and those with error values below another specified threshold are coarsened. This strategy is desirable as long as the two thresholds are carefully chosen. In our experience, we found it difficult to predict the final mesh size when choosing absolute levels for coarsening and refinement. This is particularly true when dealing with large unsteady problems where the flow features may change dramatically with time. There is a significant chance that the resulting mesh at some time step will be larger than the storage capacity of the computer.

An alternate strategy is to target a specific number of edges to coarsen and refine at each step. This targeting is based on the error indicators for the edges, but the numbers are automatically adjusted so that the total problem size remains approximately constant. For example, if the total problem size is limited to N edges, then some fraction of available edges are targeted for coarsening. Heuristics can then be used to choose the number of edges for refinement so that the problem size stays fixed at N. This way, the resulting mesh is optimized according to a predetermined problem size and a given error indicator. This strategy is used for the problems in this paper.

ADAPTIVE SCHEME

The heart of an efficient mesh-adaption scheme is the choice of a data structure. A successful data structure must contain enough information to rapidly reconstruct the mesh connectivity when nodes are added or deleted. At the same time, it must also have a reasonable memory requirement.

The data structure for our mesh-adaption algorithm is shown in Fig. 1. The most important feature of this data structure is that it consists of a series of dynamically-allocated linked lists that are implemented using the C programming language. Separate lists are maintained for the vertices, edges, elements, and boundary faces of the mesh. Each record consists of a C "structure" that contains several data items of different types (i.e. integer, pointer, etc.). This flexibility is not available in array-based programming languages such as FORTRAN.

A major advantage of the linked-list format is that items can be dynamically added and deleted. The associated computer memory can also be dynamically allocated in C. Static FORTRAN data structures require elaborate compaction and "garbage collection" procedures when items are deleted from arrays. This is not required in the present scheme.

Another notable feature of the data structure in Fig. 1 is the presence of sublists in the vertex and edge lists. Each vertex contains a sublist of all edges that share that vertex. Similarly, each edge contains a sublist of all elements that share that edge. These sublists dramatically cut down on search requirements in the code. For example, if an edge is subdivided, each element that shares that edge needs to be updated. These elements can be easily identified by traversing

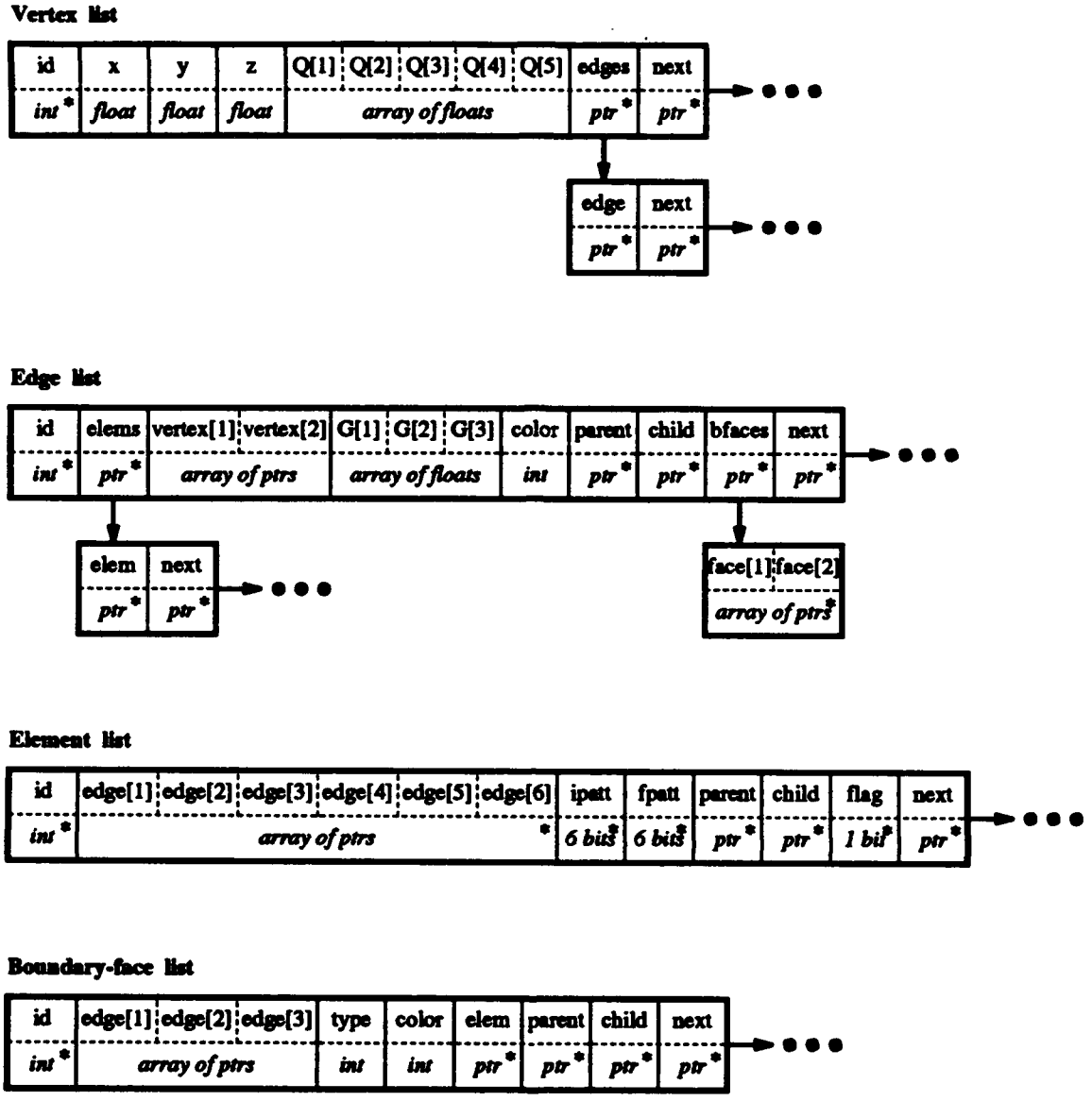


Figure 1: Record names and types for the linked-list data structure.

the element sublist for the edge.

Data structure items that are marked by a star in Fig. 1 are required exclusively by the mesh-adaption scheme. All other items are used by the flow solver. These include the five Euler variables: Q[1], Q[2], ..., Q[5], for each vertex and the three geometry components: G[1], G[2], G[3], for each edge. The "color" values are used to enable vectorization of the flow solver on the Cray Y-MP. Edges that have the same color cannot share a common vertex. The same is true for boundary faces with the same color. Edge and boundary-face colors are computed by the mesh-adaption code as soon as new edges and boundary faces are created.

The data structure is primarily based on edges of the mesh. Note that elements and boundary faces are represented as edges rather than as vertices. This feature makes it compatible with the flow solver and also enables the mesh-adaption scheme to be anisotropic. All elements in the meshes used for this paper are tetrahedra and each tetrahedron is defined by its six edges. The edges must be stored in a particular order to simplify the decision logic. An initial binary pattern (ipatt) is maintained for every element. An edge that is targeted for refinement sets its bit position in ipatt to a one in every element that shares it. These elements are located by traversing the element sublist. All other bit positions are set to zero. This means that there are 64 different subdivision patterns for each el-

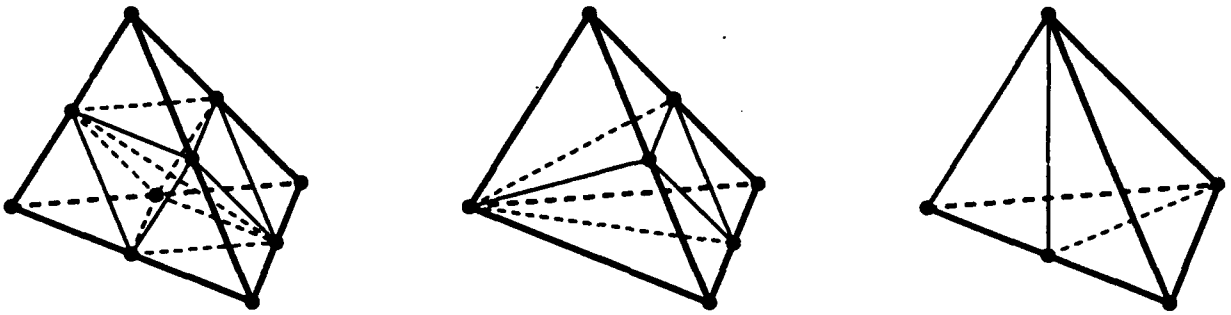


Figure 2: Three types of subdivision are permitted for each tetrahedral element.

ement although these patterns can be grouped into only a handful that are truly distinct. The advantage of these binary element-marking patterns is that they allow for extremely fast subdivision of elements. Extensive searches to find neighboring elements are not required and each element can be subdivided independently once all the edge-marking patterns are established.

Only three basic subdivision types are allowed for each element and these are illustrated in Fig. 2. The standard 1:8 isotropic subdivision is implemented by adding a new vertex at the mid-point of each of the six edges. This splits the original parent tetrahedron into eight smaller tetrahedra. As suggested by Löhner and Baum [6], the shortest inner diagonal is chosen for the refinement since it yields the least skewness in the resulting new elements. The solution vector is linearly interpolated onto each new node that is added to the mesh.

The 1:4 and 1:2 tetrahedral subdivisions are used in two ways. First, they can result because the edges of a parent tetrahedron were targeted anisotropically. Second, they are used as buffers between the 1:8 refined elements and the surrounding unrefined grid. These buffer elements are required to form a valid connectivity for the new grid system. A new point that is introduced on an edge must act as a vertex for all tetrahedra that share that edge.

Elements whose ipatt values do not match those in Fig. 2 are upgraded as shown in Fig. 3. A second binary number (fpatt) is used to store these upgraded patterns. In Fig. 3, edges 1 and 4 are initially marked for refinement. The simplest way to upgrade the binary pattern to one of those in Fig. 2 is to also mark edge 2 for refinement. Thus, the element now has three marked edges on the same face and can be subdivided 1:4. The additional marking of edge 2 may cause the fpatt values in other elements to change as well. Elements are continuously upgraded to valid subdivision patterns until none of the edges show any change. Both the initial pattern (ipatt) and the final pattern (fpatt) are stored. This way, if the element in Fig. 3 is later coarsened, edge 2 can be automatically restored, since it was not explicitly marked for refinement the first time around.

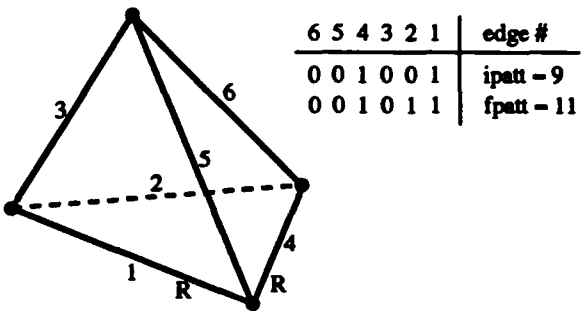


Figure 3: Initial and final edge-marking patterns for an element targeted for refinement.

Coarsening of the mesh is accomplished in four steps. First, if two sibling edges are marked for coarsening, they are replaced by their parent edge and the center vertex is removed. Next, all parent elements that share this reinstated parent edge are reinstated and their children are removed. Third, the binary marking patterns for these parent elements are adjusted to reflect the fact that some edges have been coarsened. Once these edge markings are complete, the anisotropic refinement procedure is invoked to reconnect a valid mesh.

The key factor in this mesh coarsening procedure is that the parent elements and edges are retained in the data structure. This means that child elements and edges can immediately revert to their parents after they are coarsened. They do not have to be reconstructed from scratch. Once the parents are reinstated, the mesh connectivity is reestablished using the existing element-marking patterns. There

is a memory overhead for storing the parent elements, edges, and boundary faces, but the test problems in this paper show that it is relatively small (less than 15% of the total memory requirement).

Two features of the mesh-adaption procedure remain to be described. First, there is a hierarchy between edges that are marked for coarsening and those that are marked for refinement. Even though any edge in the mesh can be refined, only those edges that have a parent can be coarsened. In other words, edges can only be coarsened if they were created by bisecting a parent edge. This means that all edges marked for coarsening must have a sibling edge. Sibling edges may be marked for coarsening (C), for refinement (R), or for no change (X). A pair of sibling edges can, therefore, only have the following combinations: R-R, R-C, R-X, C-C, C-X, and X-X. Edges that are marked R are given the highest priority and edges marked C are given the lowest. Hence, the six sibling edge patterns above are always changed to R-R, R-X, R-X, C-C, X-X, and X-X, respectively, based on the priority rules. Thus, the only way that an edge can be coarsened is if its sibling is also marked for coarsening.

A sample marking pattern for simultaneous coarsening and refinement of several elements in two dimensions is shown in Fig. 4a. Fig. 4b shows the resulting mesh after the adaption algorithm has been applied. Note that both sibling edges must be marked for coarsening in order for the edges to be removed. This has occurred in three places in Fig. 4, although the edges are ultimately deleted in only two of these instances. The node in the center of Fig. 4a is initially deleted during coarsening; however, it is added back in order to generate a valid mesh connectivity. This occurs because an element that shares this edge must be upgraded to a valid 1:4 subdivision pattern. Refinement always has the highest priority; thus, any edge that is marked for refinement will always be refined.

Note that elements have been refined anisotropically in Fig. 4. This capability is inherent in the edge-based adaption algorithm. It can be exploited in three-dimensional problems to yield efficient computational meshes for resolving directional flow features.

The final feature of the mesh-adaption algorithm concerns rules that govern when elements can be coarsened. In general, a solution-adaptive unstructured grid can contain elements that result from many different levels of subdivision. This can create problems when adjacent elements have large disparities in their subdivision levels. A two-dimensional example of this is shown in Fig. 5. Here, the large triangle is a zero-level element. Elements 3 and 7 are first-level refinements as shown in the binary tree that accompanies the figure. Elements 5 and 6 are third-level refined elements.

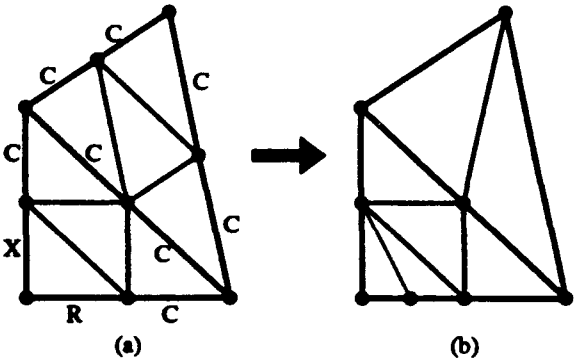


Figure 4: Coarsening and refinement are combined in this two-dimensional example to create a new mesh.

In general, elements and edges must be coarsened in an order that is reversed from the one by which they were refined. This is a fairly reasonable restriction that results from the generalized data structure in Fig. 1, but it means that only certain edges are allowed to coarsen during each mesh adaption step. The rule is that edges of non-leaf elements as well as those of leaf elements having non-leaf siblings cannot be coarsened. A leaf element is defined as one that has no children. When this rule is applied to the mesh in Fig. 5, only the edges marked with a C can be coarsened. All other edges must wait to be coarsened until these four edges are coarsened.

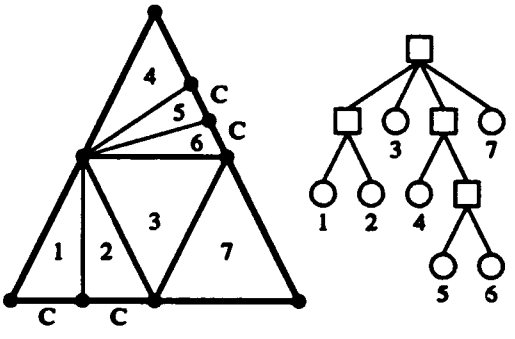


Figure 5: Only edges marked with a C can be coarsened in this two-dimensional example with multiple-level refined elements.

COMPUTED RESULTS

The new adaptive mesh algorithm was applied to two transonic flow problems in three dimensions. Even though both these cases represent steady-state problems, the adaptive-mesh algorithm is applicable

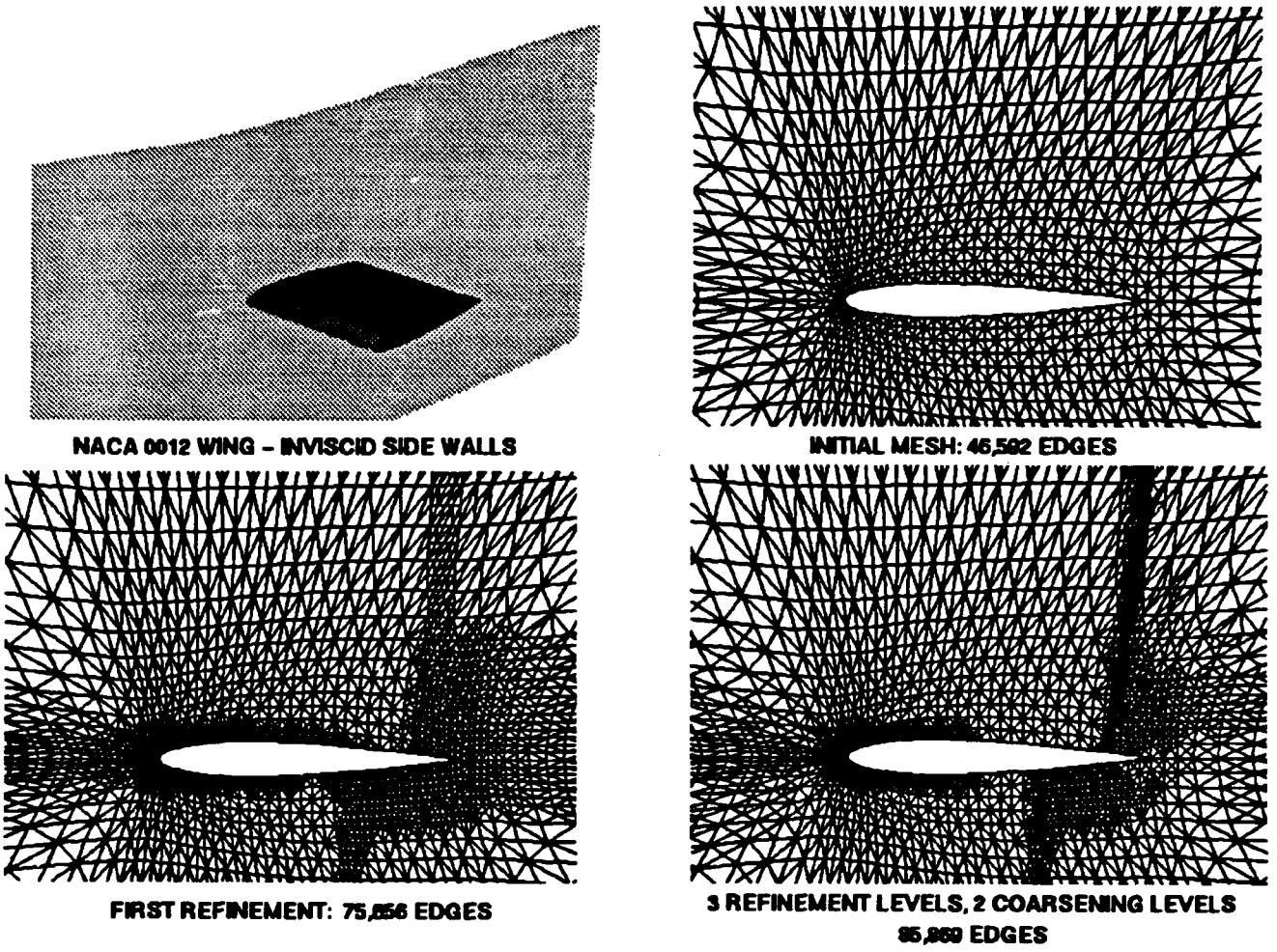


Figure 6: Several mesh adaption stages are shown for the inviscid NACA 0012 wing, $M_\infty = 0.85$, $\alpha = 1°$.

to unsteady flows as well. Steady-flow examples were chosen as the simplest test cases that fully exercise all aspects of the new mesh coarsening/refinement scheme in three dimensions.

The first test problem is a NACA 0012 wing that is mounted between two inviscid sidewalls. The advantage of this case is that although the problem is three-dimensional, the results can be easily visualized along the sidewalls and compared to existing high-resolution two-dimensional computations. Flow conditions for this case are a freestream Mach number of 0.85 and an one degree angle of attack.

The initial three-dimensional computational mesh was created from two structured-grid H-H meshes placed one chordlength apart in the spanwise direction. These structured-grid meshes were then split into tetrahedra. Each structured-grid hexahedron was divided into five tetrahedra. The initial mesh consisted of 10,556 nodes and 46,592 edges. A total of 65 nodes were located on the surface of the wing at each two-dimensional plane. The outer computational boundaries are located at approximately 14 chords above and below the wing.

Fig. 6 shows the sequence of mesh refinement and coarsening for this problem. The first mesh refinement marked 1,996 edges for refinement. This resulted in the addition of 5,270 nodes and 29,064 edges to the mesh. Two additional refinement and coarsening levels were then performed. We first targeted 7,508 edges for coarsening and 1,493 for refinement. The second targeted 4,994 edges for coarsening and 1,007 for refinement. These values were chosen only to obtain a reasonable solution for the problem. No attempt was made to optimize the adapted mesh for efficiency or the accuracy of the final computed results.

It was observed that the total problem size remains about the same if the number of targeted edges for coarsening is approximately five times the number of edges targeted for refinement. Not all of the

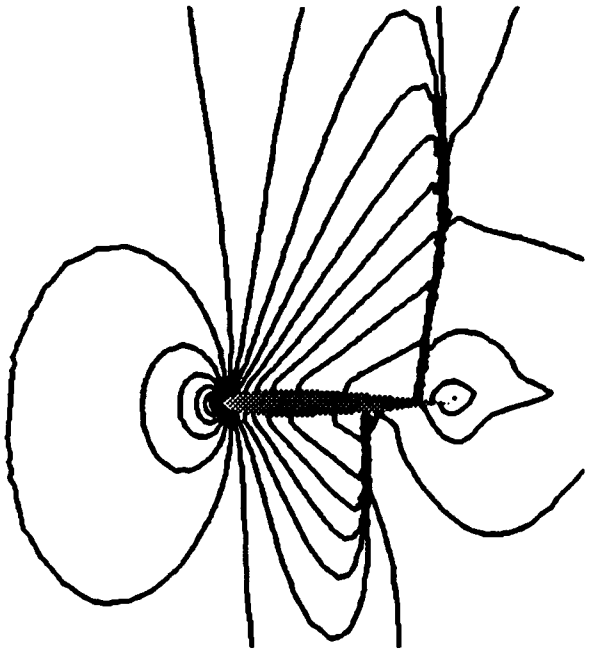


Figure 7: Computed Mach contours at the inviscid sidewalls show two distinct shocks, $M_\infty = 0.85$, $\alpha = 1°$.

edges that are targeted for coarsening will ultimately be lost. Recall that an edge can only be coarsened if its sibling is also marked for coarsening.

The final mesh in Fig. 6 has three levels of mesh refinement at the shocks and near the leading edge. This figure shows the mesh at the inviscid sidewall boundaries. Edges that are interior to the boundaries are also subdivided using our error indicator given by Eq. (2). It is clear that some of the edges from the first refinement have been redistributed in the final mesh. The result is a grid of similar size with a more optimum distribution of nodes. Although it is impossible to see in the figure, the mesh refinement is anisotropic near the shock. Only the edges that cross the shock were targeted for three levels of refinement.

The flow solver was run for approximately 750 iterations on each intermediate mesh. The fact that the new mesh starts out with the interpolated coarse-grid solution means that it converges rapidly for steady-state calculations, even on fine meshes. The final mesh has a total of 114 nodes on the wing surface at the inviscid sidewall boundaries.

It is interesting to note that there is little mesh refinement at the trailing edge. This affects the solution quality in this region and also the computed drag. The cause of this lack of mesh refinement is the choice of the error indicator. Error indicators that are based on velocity differences typically target flow regions near leading edges and shocks preferentially to trailing edges. This makes sense if you look at the relative velocity gradients in these regions. This example illustrates the fact that there is much room for improvement in the choice of a universally-effective error indicator.

Mach number contours for this case are shown in Fig. 7. These contours show excellent resolution of the two shocks both on the surface and in the far field. The stagnation point at the leading edge is also captured with high resolution. Results for computed pressure coefficient on the final mesh are presented in Fig. 8. These computed results are compared to the AGARD Euler solution N° 9 taken from [11]. This structured-grid solution used an O-mesh consisting of 320 points on the airfoil surface and 64 points normal to the surface. The outer computational boundary was located 25 chords from the airfoil surface. The results in Fig. 8 show excellent agreement between the two computed results. The slight difference in the shock locations on the lower surface may be a result of the different boundary locations and treatment for the two calculations. Although the purpose of this test case is not to optimize any particular method, it is clear that the solution-adaptive unstructured-grid calculation has a more efficient mesh distribution than its structured-grid counterpart.

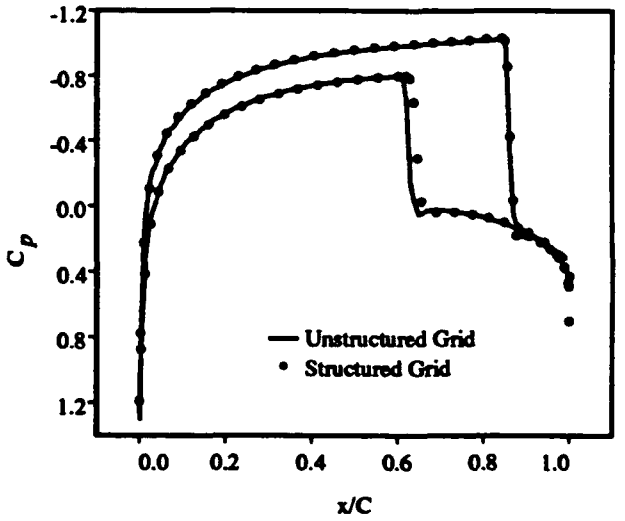


Figure 8: Computed surface pressures from the solution-adaptive unstructured-grid scheme are compared to those from a structured-grid method.

The second test case is a nonlifting rectangular helicopter rotor blade in hover. This configuration was experimentally tested by Purcell [12] who measured its acoustic performance at transonic speeds. It has an aspect ratio of 13.71 and a NACA 0012 airfoil section. High-quality structured-grid Euler calculations have been performed by Baeder [13] for this case.

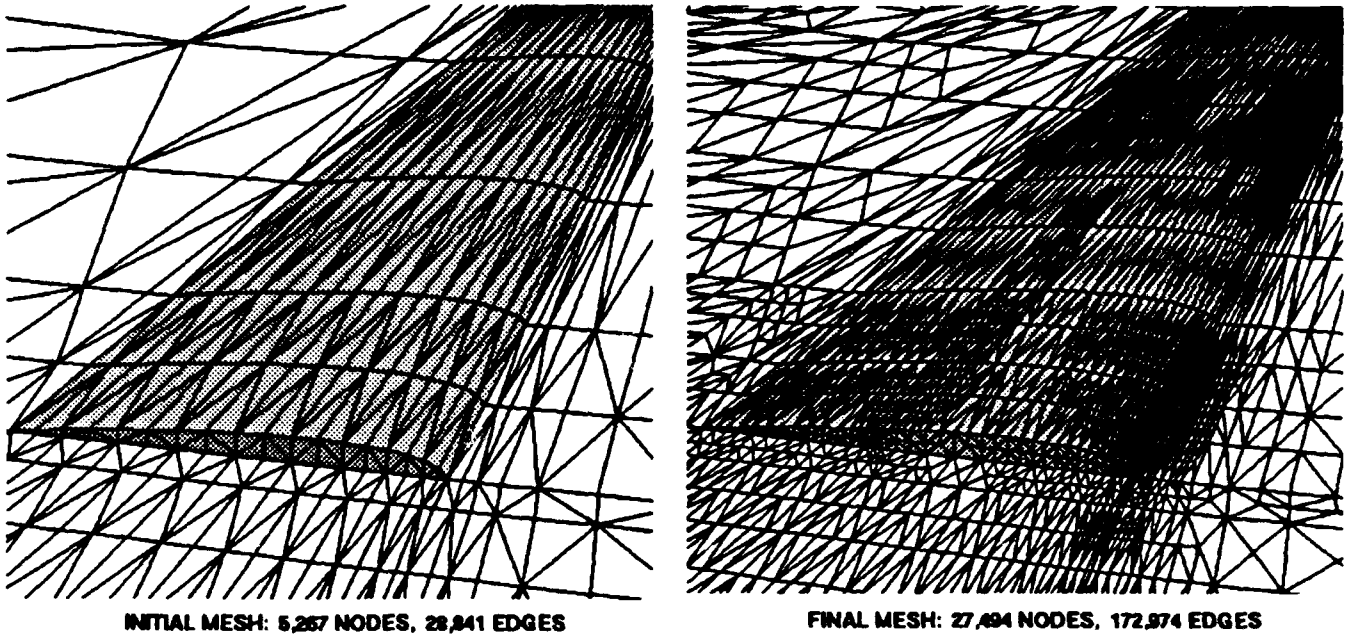


Figure 9: Initial and final meshes for the nonlifting hovering rotor case, $M_{\text{tip}} = 0.90$, AR = 13.71.

Baeder's computed results show excellent agreement with measured far-field acoustic pressures.

The unstructured-grid calculation begins with a very coarse structured C-H mesh on the upper half of the computational domain. Because the rotor is nonlifting, the problem is symmetric and only the upper half of the domain needs to be computed. The structured grid is then split into tetrahedra and the resulting mesh is shown in the left half of Fig. 9. Similar to the first test case, three levels of mesh refinement and two levels of mesh coarsening are applied to this problem. The initial refinement increases the problem size from 5,267 nodes to 25,242 nodes. The two subsequent refinement/coarsening steps each targeted approximately 25,000 edges for coarsening and 5,000 edges for refinement. Once again, no attempt was made to optimize the mesh coarsening/refinement strategy. Approximately 750 iterations for the flow solver were run between mesh adaption steps. The final mesh is shown in the right half of Fig. 9 and contains 27,494 nodes and 172,974 edges. Note that the resolution on the rotor surface is significantly improved. There are three levels of mesh refinement near the blade tip at the shock.

Fig. 10 shows the computed Mach contours on the surface of the rotor. A comparison of Figs. 9 and 10 shows that the mesh adaption has occurred primarily in the leading-edge region of the blade and along the shock near the tip. The resolution in these regions is significantly improved from the original mesh. Note that the Mach contours are not completely smooth.

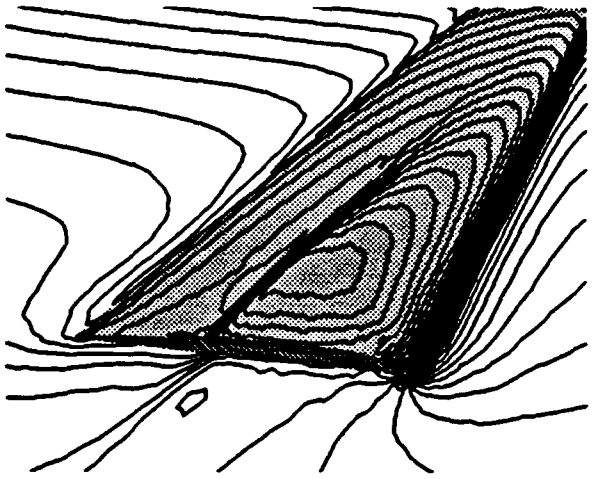


Figure 10: Computed Mach contours for the solution-adaptive rotor mesh.

This is due to a lack of smoothness in the final mesh resulting from the anisotropic mesh refinement and the fact that relatively few mesh points are used. A smoother final mesh can be realized by adjusting the error indicator and/or adding more nodes to the grid. In spite of the lack of smoothness in the surface mesh, the quality of the solution is reasonably high.

Computed surface pressures for the rotor are compared with the structured-grid results from Baeder [13] in Figs. 11 and 12. Baeder's solution uses

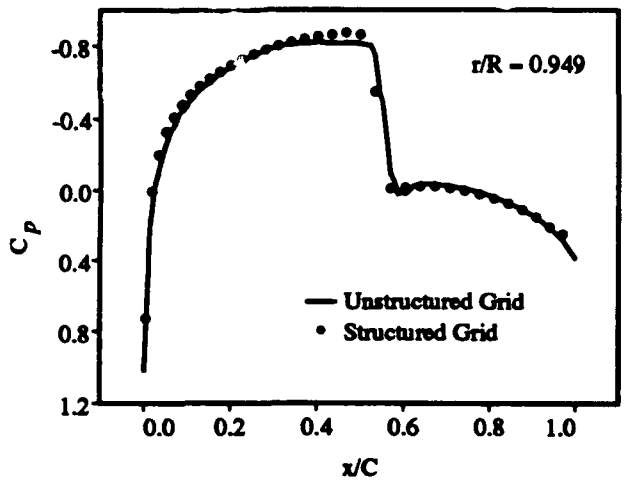


Figure 11: Computed surface pressures are compared for the structured- and unstructured-grid methods.

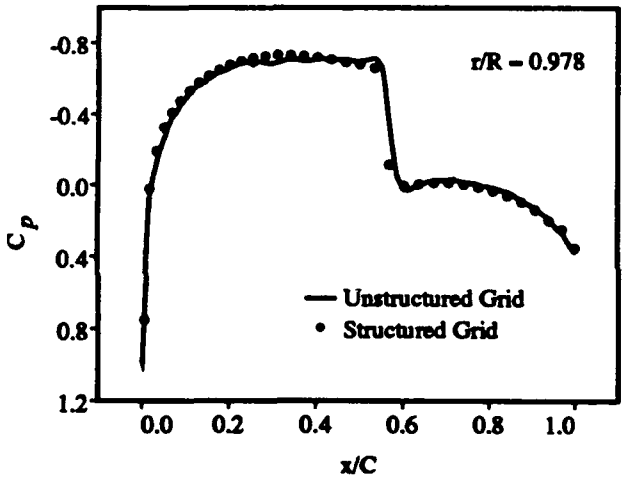


Figure 12: Computed surface pressures are compared for the structured- and unstructured-grid methods.

56,203 nodes compared to the 27,494 that are present in our unstructured-grid solution. Fig. 11 shows good agreement between the two results at the 95% spanwise location. This is particularly true of the computed shock locations. Fig. 12 shows similar good agreement closer to the tip. Here the unstructured-grid result shows some small wiggles that are the result of the lack of smoothness in the final grid. Overall, there is excellent agreement between the two computations.

COMPUTER RESOURCES

The required computer resources for the mesh-adaption algorithm depend on the problem size and the relative numbers and locations of edges that are targeted for coarsening and refinement. For this reason, it is difficult to pinpoint a rule of thumb for the computer time that is required by the mesh-adaption algorithm. However, estimates for computer memory requirements can be readily computed, and CPU times are quoted for one of the computed test cases.

Based on the data structure in Fig. 1, the total memory requirements can be computed if we make the following estimates for typical problems. The number of edges is assumed to be six times the number of nodes. The number of elements is assumed to be five times the number of nodes. Also, let d_v be the average degree of a vertex and d_e be the average number of elements that share an edge. With these assumptions, the memory that is required by the mesh-adaption algorithm per node of the computational mesh is: 12 integers, $77 + 2d_v + 12d_e$ pointers, and 65 bits. These numbers represent the additional storage requirements beyond those quantities that are directly required for the Euler flow solver. These additional items are indicated by stars in Fig. 1. This number does not include the overhead of retaining storage for the parent elements and edges. This overhead is typically small (less than 15% for the test cases in this paper). The estimate also does not include the storage requirements for the boundary-face list because this is also typically small for large three-dimensional problems.

Estimates for the CPU time can be obtained by looking at the first coarsening/refinement step for the fixed-wing test case. Recall that the mesh contained 15,826 nodes and 75,656 edges after the initial refinement. Of these edges, 7,508 were marked for coarsening and 1,493 were marked for refinement. However, only 7,026 edges were actually able to coarsen because an edge can be coarsened if and only if its sibling is also marked for coarsening. The coarsening portion of the algorithm required 3.41 CPU seconds on a Cray Y-MP computer and removed a total of 1,852 nodes and 10,079 edges. The subsequent mesh refinement required 1.87 CPU seconds and added 2,287 nodes and 13,053 edges. Thus, the combined mesh coarsening and refinement step required approximately 5.28 CPU seconds. This is only about 2.3 times the amount of CPU time required for one iteration of the Euler flow solver. None of the quoted CPU times include the time required for I/O operations.

A significant portion of the computer time for both coarsening and refinement is spent on dynamic memory allocation. The algorithm uses the C dynamic memory allocation routine called `malloc()` each time an item is added to the data structure. Similarly, the C system routine, `free()` is used to delete memory locations for items that are removed.

An alternative and much faster strategy would be to customize a memory management strategy for this specific application. Large blocks of memory could be initially allocated with the system routines, and data records could be dynamically added or deleted from these blocks with low system overhead. Such a strategy should yield significant improvements in speed.

SUMMARY AND FUTURE WORK

This paper has presented a new method for dynamic adaption of three-dimensional computational meshes. The adaption algorithm makes extensive use of linked lists and dynamic-memory allocation to rapidly reestablish the mesh connectivity when nodes are added and/or removed. The solution-adaptive scheme has been demonstrated for two sample cases and computed solutions for the Euler equations show good agreement with results from conventional structured-grid methods.

Details on computer resource requirements have been presented for the mesh-adaption scheme. The computer time and memory requirements for the method should scale efficiently for large problems. These requirements should be low enough to serve a wide range of applications on existing computers.

Future work will focus on applications of the dynamic mesh-adaption scheme to problems in helicopter aerodynamics and acoustics. The ability to locally refine and coarsen the computational mesh will enable the flow solver to capture important vortices and acoustic waves.

Another targeted milestone is the extension of the Euler code and the dynamic mesh-adaption scheme from hover to unsteady forward flight. Both the unstructured-grid Euler code and the grid adaption scheme are already set up to run unsteady calculations. The major challenge is to perform these calculations in an efficient manner. Because of CFL number limitations, the Euler code will most likely be run using an implicit solver that uses the GMRES sparse matrix algorithm and is a good candidate for testing on vector as well as on parallel machines.

Implementation of this solution-adaptive technique on distributed-memory massively parallel machines is of great interest and this area is currently being investigated. Our immediate plans call for implementing this new algorithm on the CM-5 parallel computer.

ACKNOWLEDGEMENTS

The authors would like to thank Timothy Barth and Michael Garceau for the numerous helpful discussions that have led to several improvements both in the technique itself and in the paper.

REFERENCES

[1] Chiang, Y., van Leer, B., and Powell, K. G., "Simulation of Unsteady Inviscid Flow on an Adaptively Refined Cartesian Grid," AIAA-92-0443, presented at the *AIAA 30th Aerospace Sciences Meeting*, Reno, NV, Jan. 6–9, 1992.

[2] Dannenhoffer, J. F., "Grid Adaptation for Complex Two-Dimensional Transonic Flows," Sc. D. Thesis, Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, Aug., 1987.

[3] Löhner, R., "An Adaptive Finite-Element Scheme for Transient Problems in CFD," *Computer Methods in Applied Mechanics and Engineering*, Vol. 61, pp. 323-338, 1987.

[4] Rausch, R. D., Batina, J. T., and Yang, T. Y., "Spatial Adaption of Unstructured Meshes for Unsteady Aerodynamic Flow Computations," *AIAA Journal*, Vol. 30, pp. 1243-1251, 1992.

[5] Kallinderis, Y., Parthasarathy, V., and Wu, J., "A New Euler Scheme and Adaptive Refinement/Coarsening Algorithm for Tetrahedra Grids," AIAA-92-0446, presented at the *AIAA 30th Aerospace Sciences Meeting*, Reno, NV, Jan. 6–9, 1992.

[6] Löhner, R. and Baum, J. D., "Numerical Simulation of Shock Interaction with Complex Geometry Three-Dimensional Structures using a New Adaptive H-Refinement Scheme on Unstructured Grids," AIAA-90-0700, presented at the *AIAA 28th Aerospace Sciences Meeting*, Reno, NV, Jan. 8–11, 1990.

[7] Melton, J. E., Thomas, S. D., and Cappucio, G., "Unstructured Euler Flow Solution using Hexahedral Cell Refinement," AIAA-91-0637, presented at the *AIAA 29th Aerospace Sciences Meeting*, Reno, NV, Jan. 7–10, 1991.

[8] Barth, T. J.,"A 3-D Upwind Euler Solver for Unstructured Meshes," AIAA-91-1548, presented at the *AIAA 10th Computational Fluid Dynamics Conference*, Honolulu, HI, June 24–27, 1991.

[9] Strawn, R. C. and Barth, T. J., "A Finite-Volume Euler Solver for Computing Rotary-Wing Aerodynamics on Unstructured Meshes," presented at the *48th Annual Forum of the American Helicopter Society*, Washington, DC, June 3–5, 1992, to appear in *AHS Journal*.

[10] Warren, G. P., Anderson, W. K., Thomas, J. L., and Krist, S. L., "Grid Convergence for Adaptive Methods," AIAA-91-1592, presented at the

AIAA 10th Computational Fluid Dynamics Conference," Honolulu, HI, June 24–27, 1991.

[11] *Test Cases for Inviscid Flow Field Methods - Report of Fluid Dynamics Panel Working Group 07*, AGARD-AR-211, May, 1985.

[12] Purcell, T. W., "CFD and Transonic Helicopter Sound," Paper No. 2, presented at the *14th European Rotorcraft Forum*, Milan, Italy, Sept., 1988.

[13] Baeder, J. D., "Euler Solutions to Nonlinear Acoustics of Non-Lifting Rotor Blades," presented at the *International Technical Specialists Meeting on Rotorcraft Acoustics and Rotor Fluid Dynamics*, Philadelphia, PA, Oct. 15–17, 1991.